The Report Committee for Abhibroto Mukherjee
Certifies that this is the approved version of the following report:

iMediCap App: An Android Application for Medication Assistance

APPROVED BY
SUPERVISING COMMITTEE:

Christine Julien, Supervisor

Kathleen Suzanne Barber

iMediCap App: An Android Application for Medication Assistance

by

Abhibroto Mukherjee, B.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

May 2017

Dedicated to my family.

Acknowledgements

I would like to thank my professors Dr. Christine Julien and Dr. Suzanne Barber for their guidance on this project and report. Additionally, I wish to thank my friends and co-workers on 'SmartCap: A Bluetooth low energy prototype' project, Michael Spagon and Farris Bar for their advice and hardware debugging assistance.

Abstract

iMediCap App: An Android Application for Medication Assistance

Abhibroto Mukherjee, M.S.E.

The University of Texas at Austin, 2017

Supervisor: Christine Julien

This paper proposes an Android application, iMediCap, which would be a personal assistant on the patient's smartphone. The application's ecosystem can send discreet medication reminders, record logs and provide an alert if the medication has been exposed to extreme temperatures (e.g. a hot car) that could compromise the integrity of the medication and result in undesirable side effects or an ineffective treatment. In addition, if the treatment requires multiple medications, iMediCap-App can propose the optimum schedule. Furthermore, it provides drug related information and adherence related analytics on the app and through an interactive web-interface respectively.

Table of Contents

List of Tables viii

List of Figures ix

Chapter 1: Introduction 1

1.1 Mobile Device Interface 2

1.2 Mobile Application Implementation 4

1.2.A BLE Enabled User Interaction 5

1.2.B Non- BLE Enabled User Interaction 6

1.3 Database Architecture and Design 7

Chapter 2: Analytics and Tracking Portal 11

Chapter 3: Security Improvements 13

3.1 Protecting Sensitive Information 13

3.2 Session Keys and Cookies 14

3.3 CSRF Protection 16

3.4 Security Bugs in SmartCap 1.0 and Fixes 16

Chapter 4: Performance Analysis 18

4.1 Load and Stress Test Results 19

Chapter 5: Discussion and Conclusions 25

5.1 Performance Results and Limitations 25

5.2 Future Work 26

Appendix 27

References 33

List of Tables

Table 1.1: SQL vs MongoDB Concepts and Terminology.....8

Table 4.1: Load Test summary.....19

Table 4.2: Stress Test summary 23

List of Figures

Figure 1.1: SLTB001 by Silicon Labs. 2

Figure 1.2 UART Debugging Console 4

Figure 1.3: Login Activity 5

Figure 1.4: Home Activity 5

Figure 1.5: Multiple SmartCap enabled 6

Figure 1.6: Data from BLE module. 6

Figure 1.7: Alternate Activity log for non-BLE Users. 7

Figure 1.8: Database Design and Architecture for iMediCap.....9

Figure 2.1: Analytics web-portal 11

Figure 3.1: (a) SmartCap 1.0 password was stored in plain text (b) iMediCap password is encrypted using Bcrypt. 13

Figure 3.3 Android activity lifecycle [10] 17

Figure 4.1 Response Time Graph for Load Test.....20

Figure 4.2 (a), (b) and (c) Increase in CPU and Memory utilization with Load ...22

Figure 4.3 Stress Test Ramp up 22

Figure 4.4 Stress Test Response time 23

Figure 4.5 Stress Test Server vitals.....24

Figure A.1 Components of SmartCap System 31

Chapter 1: Introduction

Lack of adherence to the prescribed schedule or taking medication with compromised integrity can have life-threatening side effects. Replacement of compromised medication is costly and is not covered by most health care plans. Since, smartphones and tablet computers have become increasingly popular, the market for personal assistance apps continues to grow. As part of course curriculum, Farris Bar, Michael Spagon and Abhibroto Mukherjee came up with a course project idea in fall 2016, 'SmartCap: A Bluetooth low energy prototype'. This was a proof-of-concept course project to provide solution to ensure medical adherence.

The SmartCap app system provided the fundamental framework. It consisted of the following:

- Android application with features to add prescriptions, medication reminders, track compliance on daily basis
- BLE- module with sensors for temperature, humidity and on/off buttons for monitoring the storage temperature and providing an alert if the medication has been exposed to extreme temperatures or humidity. The app also kept track of opening and closing of the cap.

Although, SmartCap 1.0 had lot of great features to help its users adhere to their medication by providing tracking and reminders. It lacked any robust security measure to protect user's PR information. Also, as we saw above, the system required the users to obtain a BLE kit enabled cap to keep track the medication timings. Which meant there was no way for a user, who doesn't desire to obtain such BLE enabled caps, log their medication compliance. All these challenges were addressed in the iMediCap version and which also comes with additional features such as analytics web portal.

The iMediCap app systems is hosted on AWS (Amazon Web Services EC2), which significantly improves scalability and performance by using load balancers. It also uses secure authorization tokens to secure all the RESTful API access and communication between server and clients. In addition, the new system includes added

features, a redesigned android app, and an analytics web portal where user to track progress and compliance trends.

1.1 Mobile Device Interface

As part of the SmartCap (1.0) project the mobile device interface with the BLE module was developed. The BLE module on the periphery consisted of temperature and humidity sensors, and an on-off switch to replicate cap opening action. We used the Thunderbolt Sense SLTB001A by Silicon Labs for the sensors.

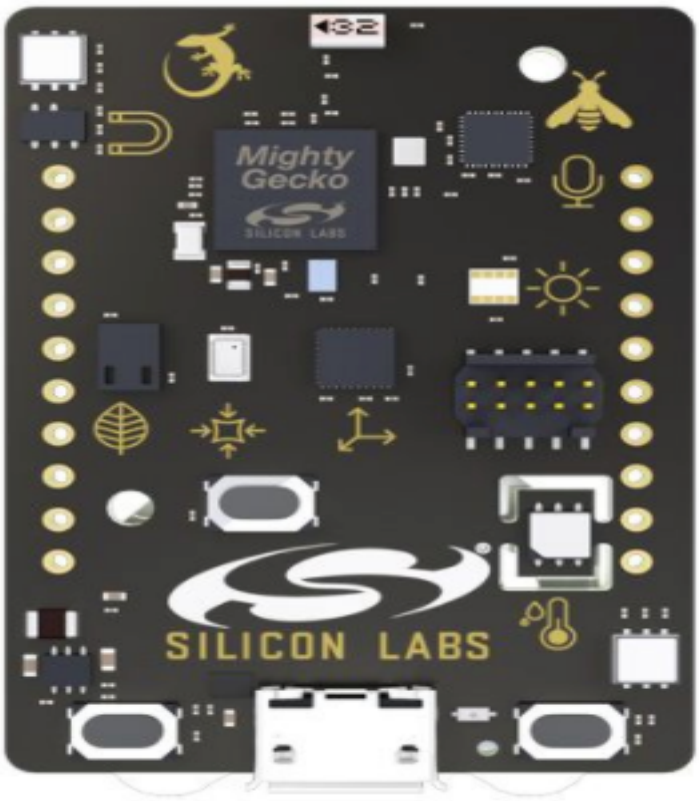


Figure 1.1: SLTB001 by Silicon Labs.

The temperature and humidity sensors are polled every 5 minutes. However, an event (a trigger that causes a data frame to be stored in the cyclical buffer) is not generated if the temperature and humidity do not change by at least 5% of the previously

written value; the value is just discarded. Our justification for this is that the device will usually be in a static environment. Events are generated by the SmartCap when one of four things happens:

- Open event (type 0)
- Close event (type 1)
- Temperature event (type 2)
- Humidity event (type 3)

A UART console was used to display debugging information. Having this console open while transferring data to the smartphone played an important role in our debugging process. When the firmware sends the log information to the application, it waits for an acknowledgement flag before clearing the data from the cyclical buffer; data should only be deleted on successful transmission. GATT is an acronym for the Generic Attribute Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics. We implemented our own GATT service for transmitting our event logs.

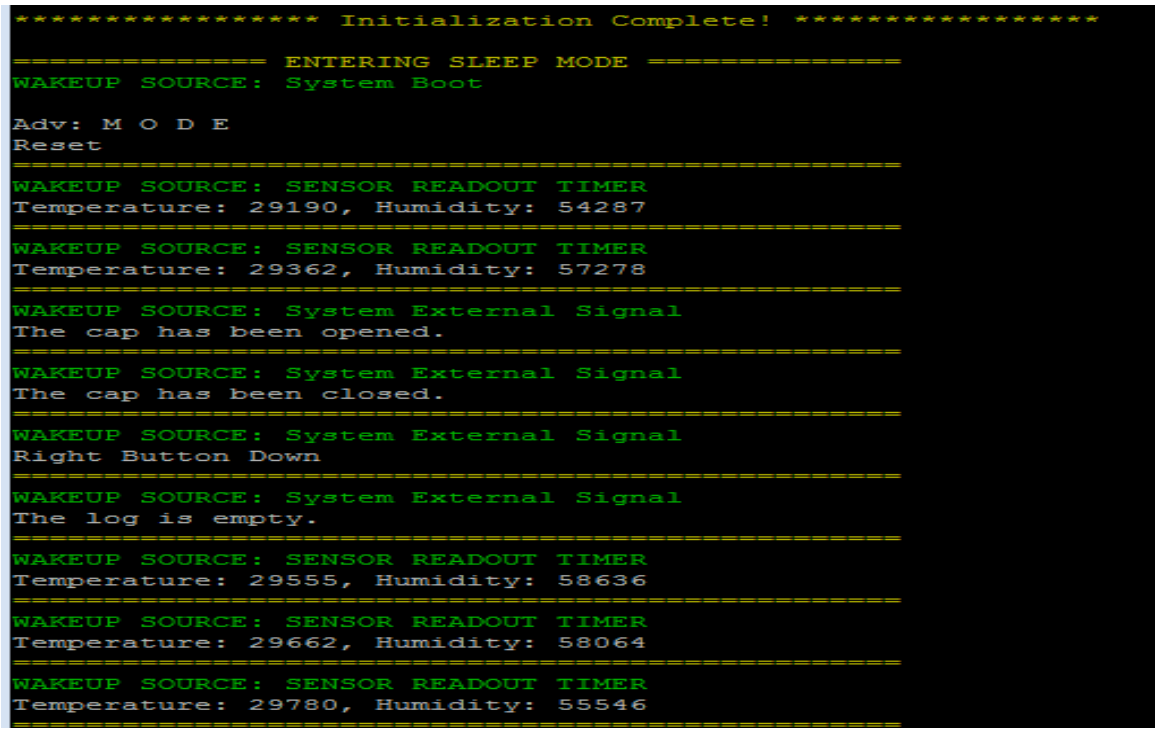


Figure 1.2 UART Debugging Console

1.2 Mobile Application Implementation

The iMediCap android application is broken down into multiple activities. For example, the iMediCap app's starting point is the login activity shown in Figure 1.3. Here the user can login *or* signup. After a user logs in and the credentials are validated by the server, the application navigates to the home activity shown in Figure 1.4.

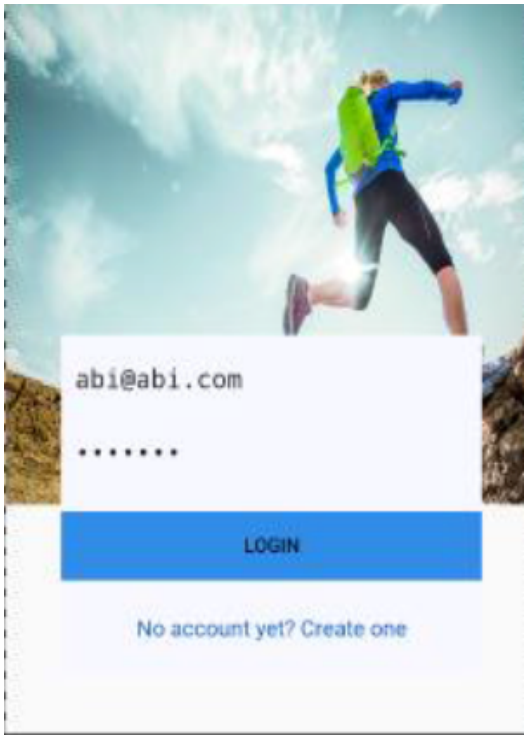


Figure 1.3: Login Activity

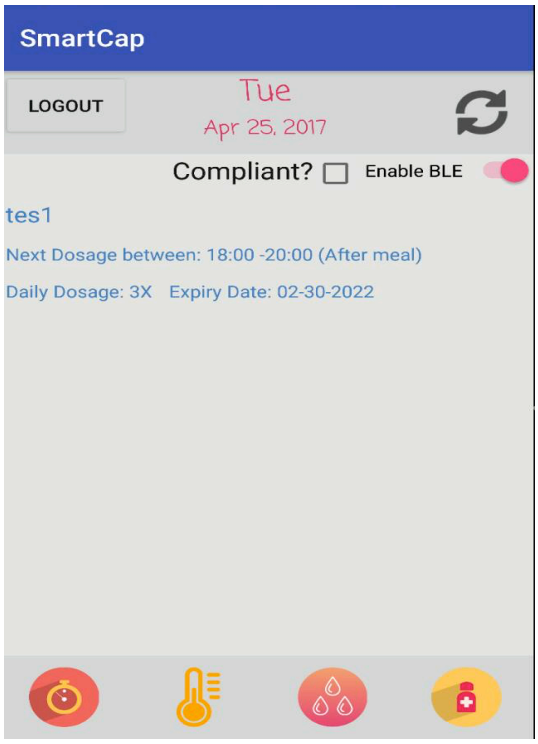


Figure 1.4: Home Activity

1.2.A BLE Enabled User Interaction

Users who acquired the BLE module enables the 'Enable BLE' toggle on the home activity. This action turns on the Bluetooth on the user's device. The home activity then displays all the BLE devices within range of 10 feet. As seen in Figure 1.5, all the available prescription caps are depicted as Item 1, Item 2, …. When the user taps on a device listed in the home activity, a BLE connection is established with the corresponding SmartCap hardware.

When a connection is established the BLE Update activity is opened (Figure 1.6). The medical adherence log can then be read from the SmartCap's cyclical buffer when the update button is pressed. The log is uploaded to the server for permanent storage when the upload button is pressed. The Figure 1.6 displays only the event template for each logged event. To see the timestamp, type of event (cap open, cap closed, temperature), and data (temperature value) user can tap on the required event.

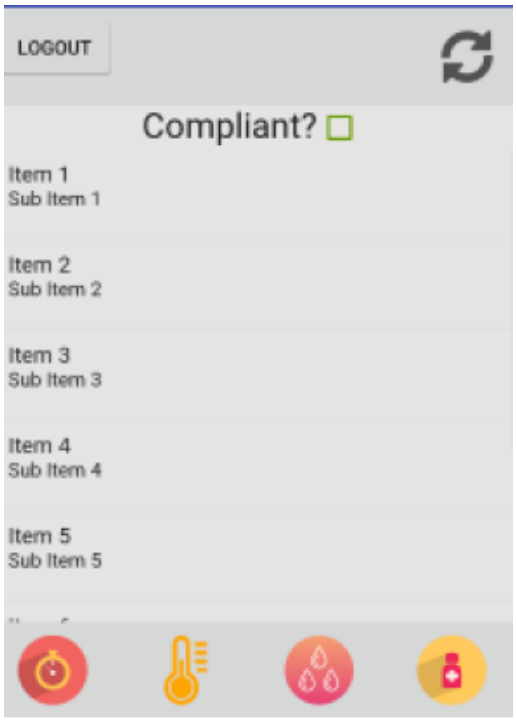


Figure 1.5: Multiple SmartCap enabled

Figure 1.6: Data from BLE module.

1.2.B Non- BLE Enabled User Interaction

For users who do not wish to acquire a SmartCap (BLE kit Cap), the iMediCap app provides an alternate feature to log activities. User can log the time of taking the medication by tapping on the medicine name on the home screen and then tapping on the 'Took Medication' option displayed below in Figure 1.7. This feature was added in the 2.0 version to improve the usability and marketability of the App.

The temperature and humidity activities, which are accessed by pressing the thermometer icon and water droplet icon at the bottom of the home activity respectively, show a red alert if the humidity or temperature range exceeds a pre-specified normal range at any time, but otherwise display green for approval. These features however need the user to have at least one BLE kit cap configured. Text message SMS alerts are sent to the user's phone as reminders.

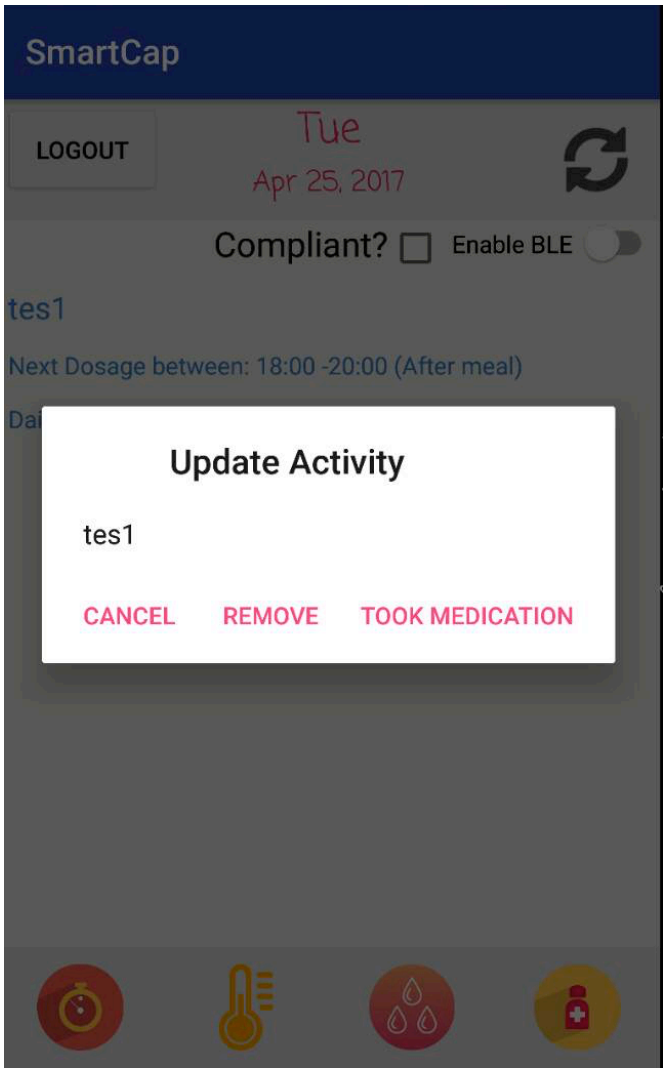


Figure 1.7: Alternate Activity log for non-BLE Users.

1.3 Database Architecture and Design

The back-end of the iMediCap is developed based on ExpressJS framework with NodeJS and NoSQL database MongoDB [5]. MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and the data structure can be changed over time. MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. The table below provides a quick comparison between SQL and NoSQL database terms and concepts.

SQL Terms and Concepts	MongoDB Terms and Concept
Database	Database
Table	Collection
Row	Document
Column	Field
Table Joins	$lookup

Table 1.1: SQL vs MongoDB Concepts and Terminology

The iMediCap application consists of five persistent collections, Patient, Users, Events, Temperature, and Compliance. The patient collection contains all the drug related information and user's collection comprises of _id, username, name, password, and phone#. This separation in collections helps us keeping the patient's medical records secure. The diagram below shows the schema, composition and relation between the collections.

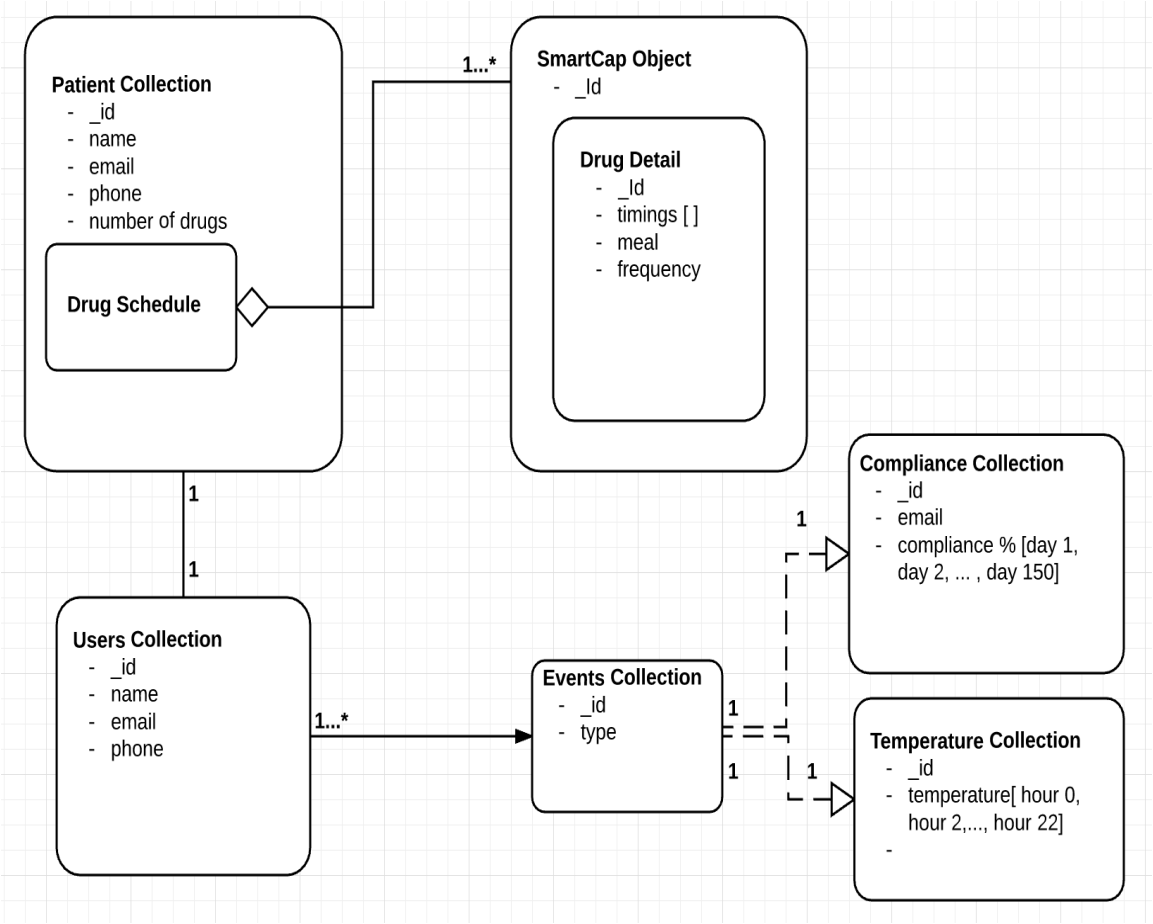


Figure 1.8: Database Design and Architecture for iMediCap

Here's an example of the record stored in patient collection for which we saw the Login and Home activity in Figures 1.7 and 1.8 respectively

```
{"_id" : ObjectId("58d326447cc4782286ca2250"), "name" : "Abhi", "email" : "abi@abi.com",
"password" : "$2a$10$3OEgqwONLjiUYH5k8EG/m.ChDzUYA7YnnAuC1hGSwDj51RTCdZZGW", "phone" : "6598",
"smartcap0" : [ "62829", "tes1", "3X", "After", "02/30/2022" ], "number_of_drugs" : "1" }
```

Figure 1.9: Patient record example

In order to explore the possibilities of a smartphone based personal assistance system that helps patients improving his/her medication adherence, this paper proposes an Android application, iMediCap. The iMediCap system would provide the users features such as daily medication tracking, temperature, humidity monitoring, web-portal to understand adherence trend etc. The iMediCap system provides great security features

to protect user's medical and other personal information using modern encryption techniques. In addition, the system is highly scalable and was developed keeping performance in mind. As we will see in later chapters it is able to handle multiple concurrent requests with high efficiency and at low error rates.

Chapter 2: Analytics and Tracking Portal

iMediCap also comes with a web-based analytics and tracking portal that helps the user to quickly understand his or her compliance trends and track progress. The user can also see the temperature trend so that he or she can take appropriate action if required to ensure the drug integrity.

The analytics portal computes the average compliance percentage based on the number of prescription drugs for the user and number of times the user logged the drug was taken within the time frame suggested in the app. This information is conveyed to the user using easy to comprehend charts. Below is the screenshot for a test user for whom we simulated temperature ranges and compliance data. The overall compliance trend graph captures data over a period of 150 days, the target is to encourage the user to reach 100% compliance consistently.

Figure 2.1: Analytics web-portal

Compliance score or percentage calculation is explained by using the following example: Let patient 'A' has three medications, m1, m2 and m3. Two of which, m1 and m2 needs to be consumed three times and m3 twice every day. At the end-of-day the system looks-up the events collection and calculates daily percentage based on number of times patient correctly takes the medication. If for example patient A, took the medication m1 and m2 only twice on a day, the compliance percentage falls to 66%. Now, if m3 was taken twice (as prescribed), this would mean for the day Patient A's compliance score is 83%. This data is then stored in the compliance collection in database.

The graphs and donut charts are powered by CanvasJS [3] which provides an easy JavaScript and HTML5 based APIs to configure and visualize data.

The analytics web portal is hosted on the same AWS server to reduce network calls and latencies. The temperature and medication intake logs are maintained in 'events' collection and using Mongodb-Cron [4], and end-of-day stored procedure is run to push the data to another collection called compliance.

In the current setting the information is accessible only to the individual user. However, in future such a system can be used by pharmacies or other stakeholders service corporations at a macro level to understand the behavior and improvement of iMediCap owners in terms of medication compliance. This can be used to incentivize patients, by reducing health insurance premiums and/or medication costs.

Chapter 3: Security Improvements

3.1 Protecting Sensitive Information

The SmartCap 1.0 used a NoSQL database (MongoDB) to store user data and had no encryption or protection feature for sensitive data such as password. The iMediCap protects such information by hashing using Bcrypt before storing in the database. Bcrypt is a password hashing function based on blowfish cipher.

```
> db.users.find({name:"Abhi"})
{ "_id" : ObjectId("58cdec60ced26377fc37d1f7"), "name" : "Abhi", "email" : "abi@
abi.com", "password" : "test", "phone" : "6598" }
```

```
> db.users.find({name: "Abhi"})
{ "_id" : ObjectId("58ec700ee06d874ce900d6da"), "name" : "Abhi", "email" : "abi@abi.com", "p
assword" : "$2a$10$3OEgqwONLjiUYH5k8EG/m.ChDzUYA7YnnAuC1hGSwDj51RTCdZZGW", "phone" : "6598"
}
```

Figure 3.1: (a) SmartCap 1.0 password was stored in plain text (b) iMediCap password is encrypted using Bcrypt.

The prefix "$2a$" in the hash string indicates that hash string is a Bcrypt hash in modular crypt format[1]. The rest of the hash string includes the cost parameter, a 128-bit salt (base-64 encoded as 22 characters), and 184 bits of the resulting hash value (base-64 encoded as 31 characters)[2]. The cost parameter specifies a key expansion iteration count as a power of two, which is an input to the crypt algorithm. For example, the below shadow password:

"$2a$10$3OEgqwONLjiUYH5k8EG/m.ChDzUYA7YnnAuC1hGSwDj51RTCdZZGW"

specifies a cost parameter of 10, indicating 2^{10} key expansion rounds. The salt is "*3OEgqwONLjiUYH5k8EG/m.ChDzUYA7*", and the resulting hash is "*YnnAuC1hGSwDj51RTCdZZGW*". The user's password itself is never stored.

3.2 SESSION KEYS AND COOKIES

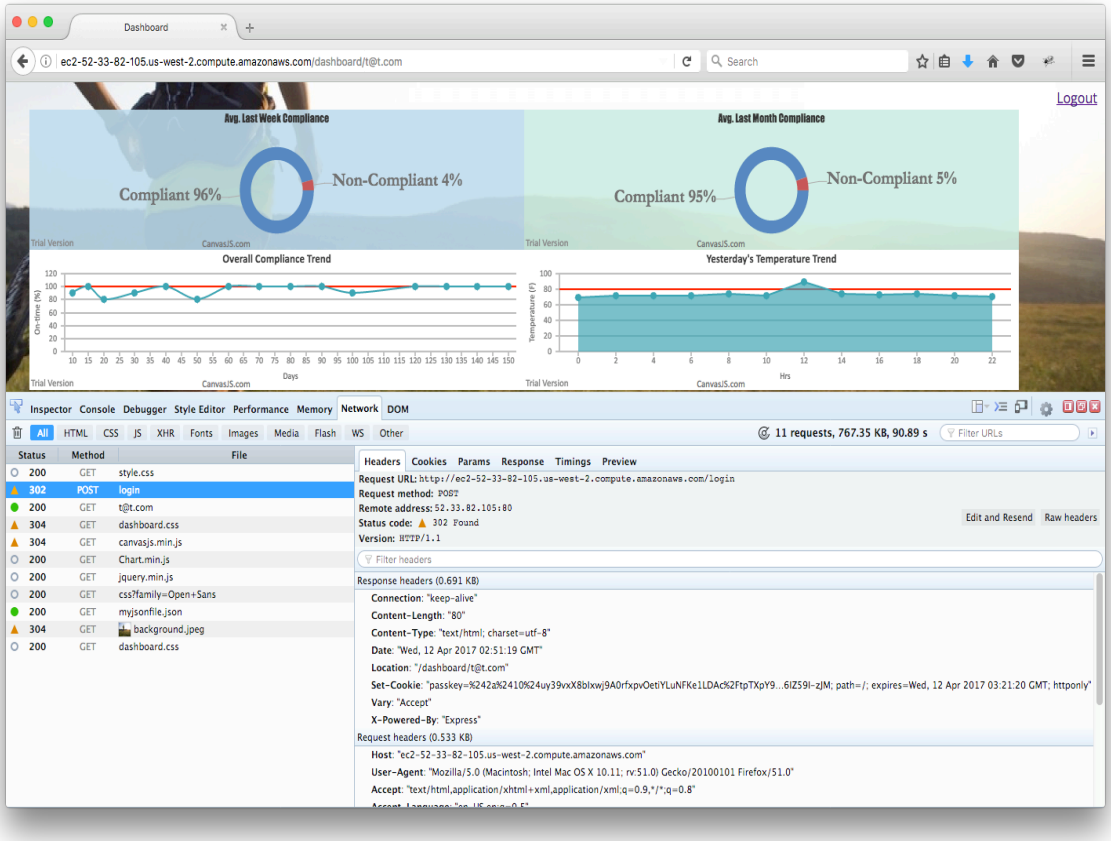
To protect hackers by-passing authorization and using session riding attacks from accessing sensitive information, it is common that web applications need to maintain session state: i.e., some state whose lifetime is limited to the current session, and that is bound to the current session.

One method of maintaining session state between a client and a server is to use a cookie to hold session information. The server packages the session key for a client in a cookie and sends it to the client's browser. For each new request, the browser re-identifies itself by sending the cookie (with the session key) back to the server.

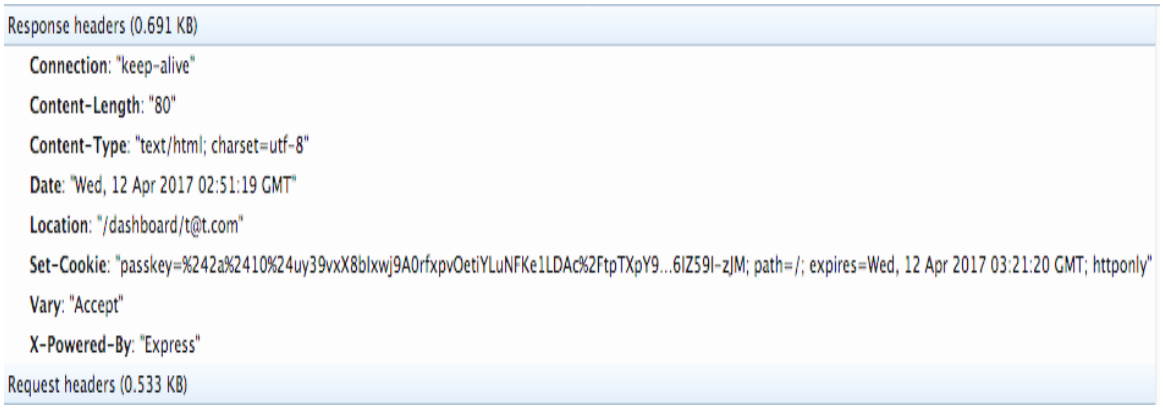
The session cookie is a server-specific cookie that cannot be passed to any machine other than the one that generated the cookie. The session cookie allows the browser to re-identify itself to the single, unique server to which the client had previously authenticated.

The session key stored in the session cookie contains only a random number identifier (“passkey”) that is used to index the server's session cache. There is no other information exposed in the session cookie.

iMediCap uses these principles by incorporating “passkey” validation on the server side to provide access to all the underlying REST API. When a user tries to login using the registered email and password, a passkey is generated at the server, and sent as part of response header. This is stored in the client cookies. From that point on all the communication between the client and the server requires the verification of this “passkey”. This encrypted passkey has attributes that makes it even stronger, such as it expires every 90 seconds, and it can only be sent over HTTP. This ensures that only one session is created and that it is over a web medium. Once the user logs out, the session and all the cookies are destroyed. This is shown in the figure 3.2 below.



(a)



(b)

Figure 3.2 (a) and (b): Network calls and Session Cookie “passkey” with attributes

3.3 CSRF PROTECTION

Since the iMediCap system depends on various user inputs such as user registration, login and entering prescription information, such requests are handled by server using POST end-points. Having such REST endpoints makes the system vulnerable to Cross-Site Request Forgery (CSRF) attacks. CSRF [7] is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. In the context of iMediCap, a successful CSRF attack can force the user to perform state changing requests like altering medication, schedules, etc.

iMediCap uses CSRF token with all form submissions (POST and PUT) to mitigate such attacks. At the implementation level this is ExpressJS provides library called 'csrf' [8] to generate these tokens and help validate the tokens at the server side.

3.4 SECURITY BUGS IN SMARTCAP 1.0 AND FIXES

On the Android application side, one of the major bugs was, if the user logged out of application and pressed the "back" key on the phone, it would display the home activity of the user. This was mainly due to inappropriate handling of activities.

Figure 3.3 shows the android application activity life cycle, which provides useful information on how activities should be managed based on a user's navigation through the application. By fixing the activity life, i.e. fixing the flow from *start (), onResume ()* and finally *finish ()*, this bug was fixed. Below is a code snippet showing how to end an activity when a user starts a new activity [11].

```
Intent intent = new Intent(this, NextActivity.class);
startActivity(intent);
finish();
```

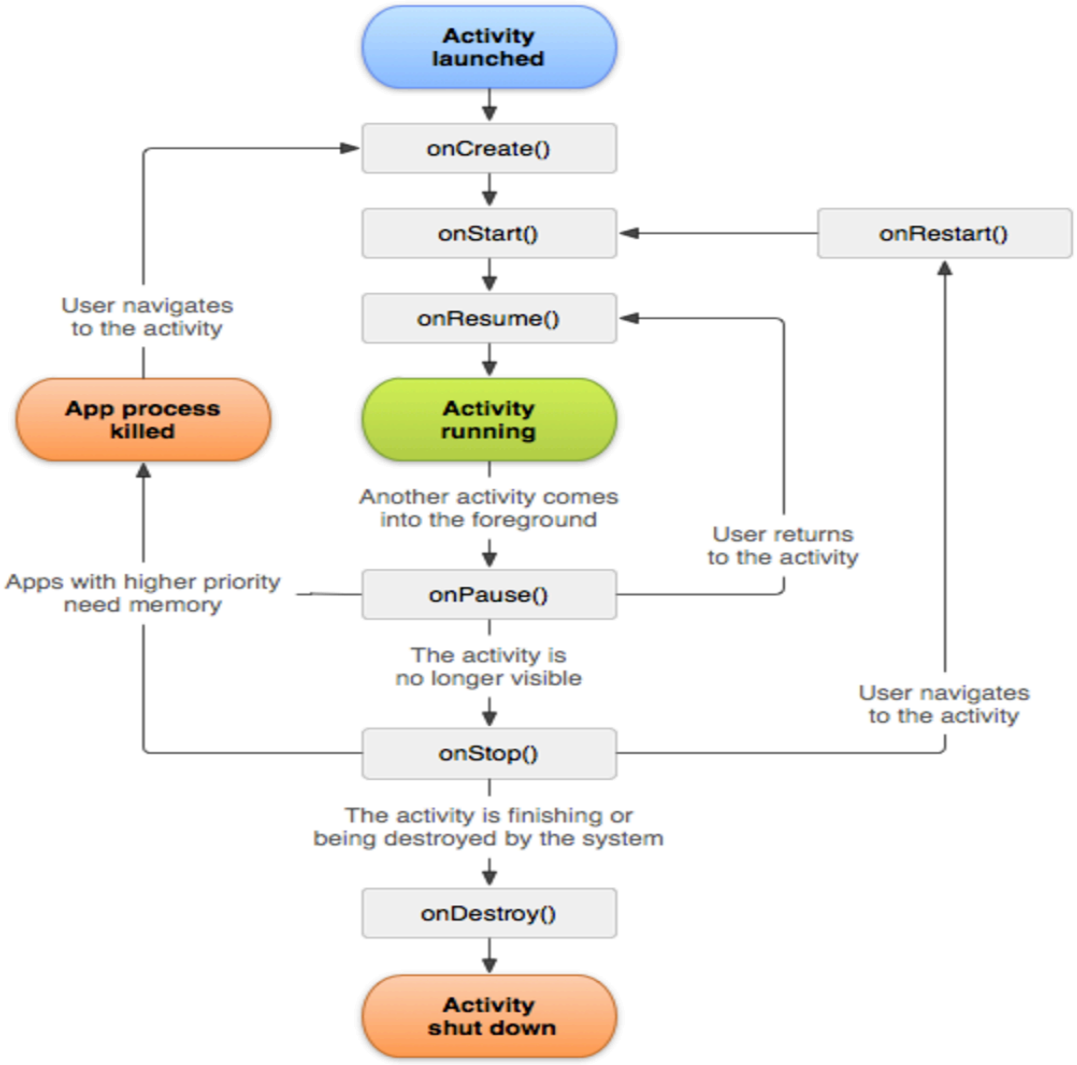


Figure 3.3 Android activity lifecycle [10]

Another bug on the android application side was that there were no session timeouts and the user could go back to their home screen (without having to login) even after hours of inactivity on the app, if they kept the application running in the background. This was prevented by introducing sessions as described in the previous section.

Chapter 4: Performance Analysis

The iMediCap is hosted on an AWS EC2 instance. Since it is not yet ready for public use through play store, the server does not contain a load balancer that automatically scales as the requests to the server increases. However, conducting some load and stress testing on the server provided us information which would be very useful to understand the infrastructure requirements that would be necessary to completely scale such a product and make it a highly available application.

The application was tested under both load and stress test. For the load test, we ramped up to 500 concurrent users who enter their login and password and view the dashboard /home activity. The test ran for 4 mins, which included a sudden surge in user activity and then a slow ramp down after the dashboard page is loaded.

For the stress test, we started with 30 users and then added 30 users every 20 seconds, until 200 concurrent users existed, then hold this load for 30 seconds and then ramp down at 50 users every 10 seconds. The 30 second hold time is a fair assumption that a user would on an average spend 30 seconds on the app or the analytics portal to get all the information he/she needs. Figure 4.3 shows the ramp up and ramp down pattern for the stress test. For performance testing the application we used Apache JMeter™, an open source software, a 100% pure Java application designed to load test functional behavior and measure performance.

4.1 LOAD AND STRESS TEST RESULTS

Table 2(b) shows the summary report of the test. Under samples we can see the number of concurrent users, 505. The throughput was around 46 RPM. The important thing to observe here is the low error rate, which stood at 0.4%. This means that, although the response to the request was considerably slow, the requests were completed successfully 99.6% of the time. This brings a very interesting observation: that by using more CPU resources and distributing the load across servers, we can significantly reduce the response time and increase our throughput. AWS provides a great solution out of the box such as Elastic Load Balancing [12]. However, the low error rate is a very encouraging.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
Login	505	144740	137526	182415	184537	187929	13988	232961	0.40%	0.8	6.1
TOTAL	505	144740	137526	182415	184537	187929	13988	232961	0.40%	0.8	6.1

(a)

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login	505	144740	13988	232961	32949.31	0.40%	46.1/min	6.08	8096.5
TOTAL	505	144740	13988	232961	32949.31	0.40%	46.1/min	6.08	8096.5

(b)

Table 4.1: Load Test summary

Figure 4.1 shows the response time trend for the 500-concurrent users. As expected, as the load on the server increased, the lag in the response time and page load time increased as well.

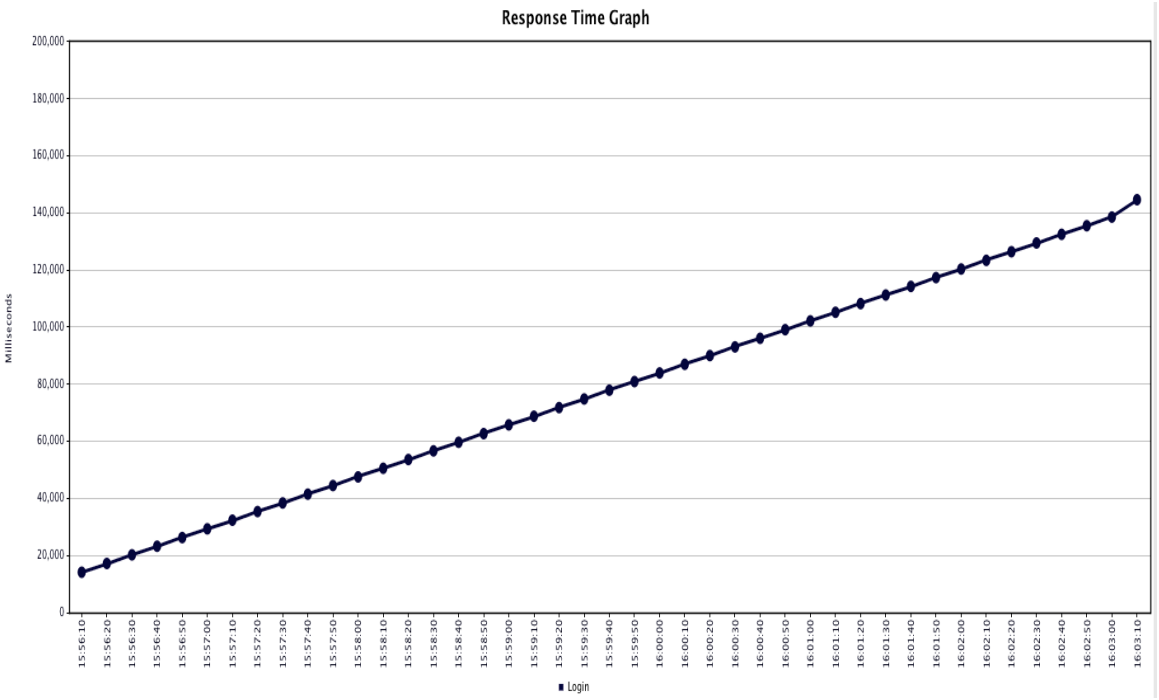
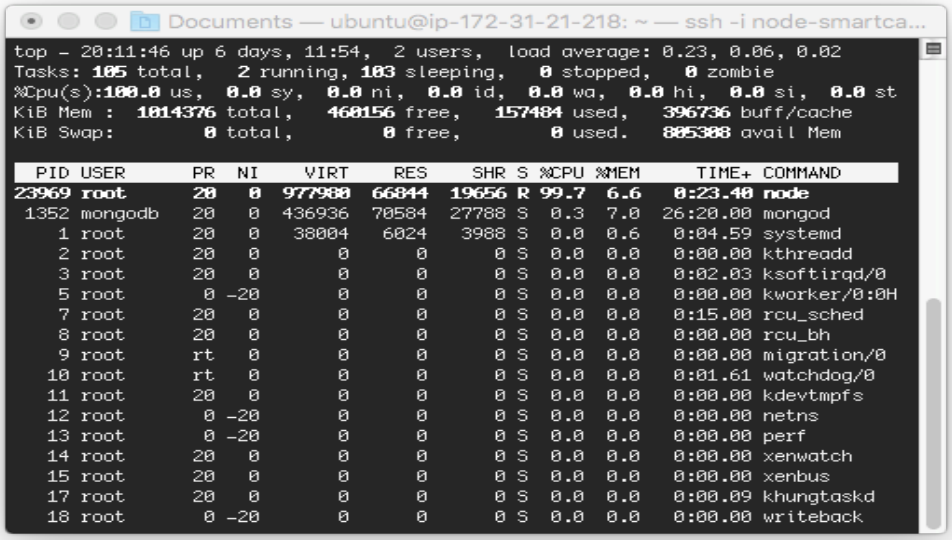
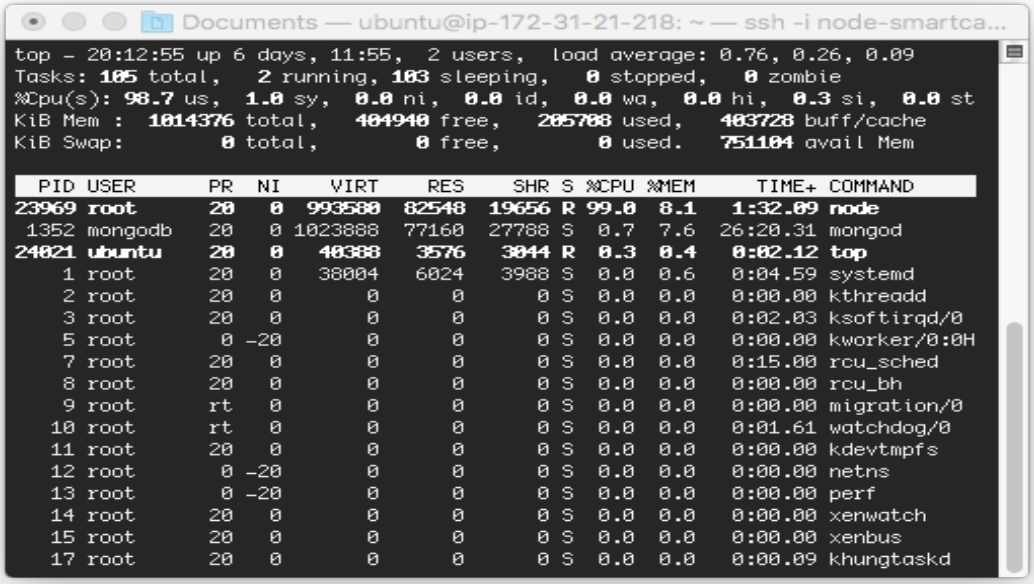


Figure 4.1 Response Time Graph for Load Test

Monitoring the server with the Ubuntu command ‘top’ we could track the server utilization. As we see in the below figures, the CPU utilization shoots up to 99% for the maximum load. The server runs on Ubuntu-xenial-16.04-amd64 server with 8 GB memory size. Figures 4.2 (a), (b) and (c) below shows the increase in both CPU and memory I/O utilization with the increase in load.

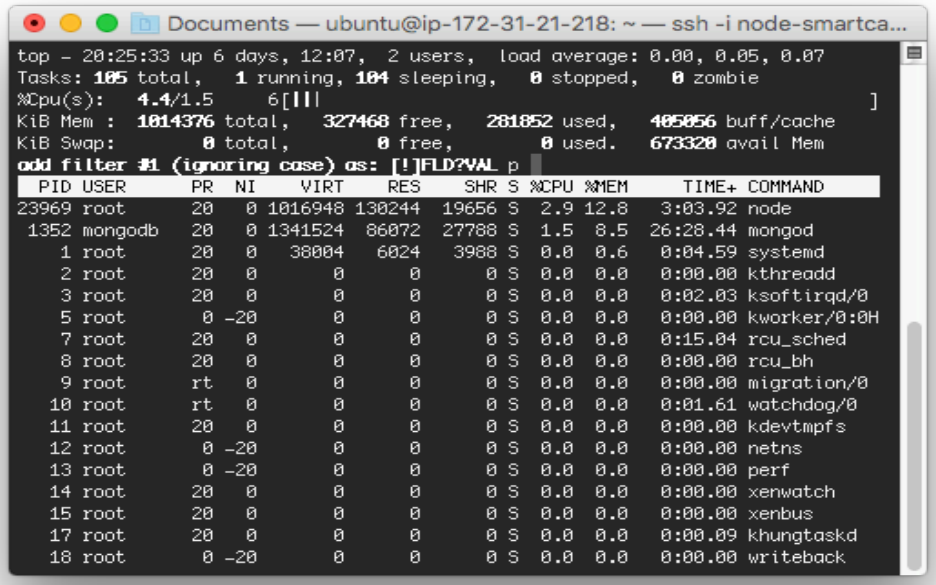


(a)



(b)

Figure 4.2: continued next page



(C)

Figure 4.2 (a), (b) and (c) Increase in CPU and Memory utilization with Load

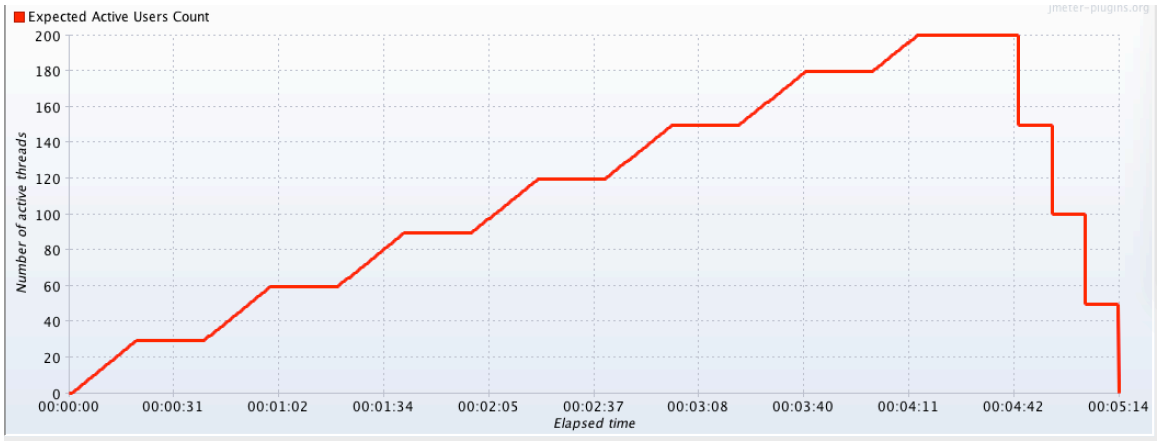


Figure 4.3 Stress Test Ramp up

Table 3 shows the summary report of the test. Under samples we can see the number of concurrent users, 530. The throughput was significantly better than what was observed under load test at 1.5 RPS. Also, there was no significant increase in the error rate than what we observed under the load test.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login	530	76229	16606	185057	24775.87	0.57%	1.5/sec	11.40	7550.9
TOTAL	530	76229	16606	185057	24775.87	0.57%	1.5/sec	11.40	7550.9

Table 4.2: Stress Test summary

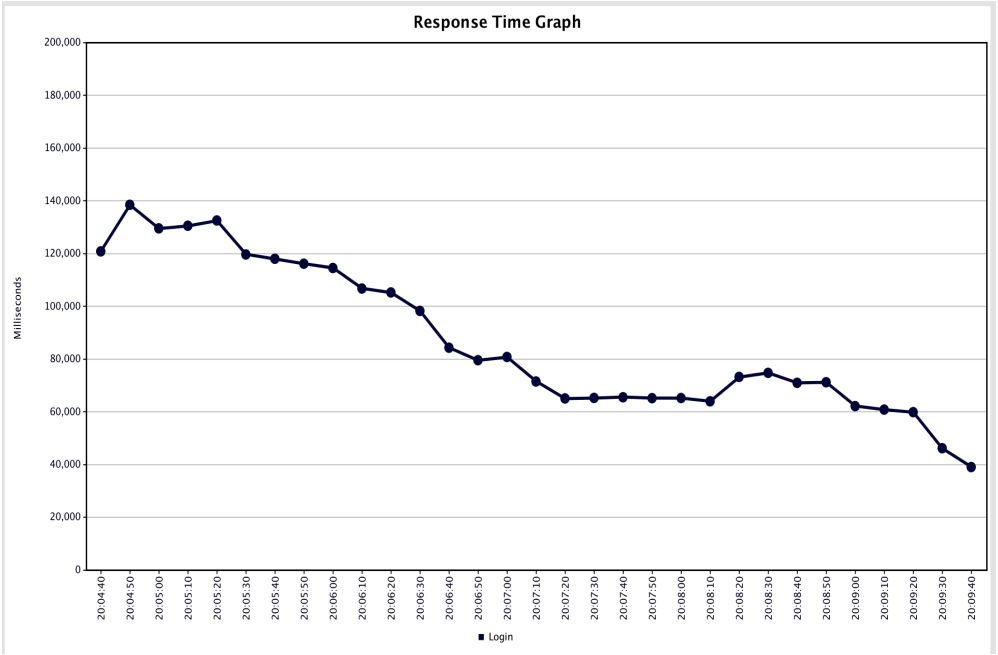


Figure 4.4 Stress Test Response time

As expected the memory I/O usage was lower than the load tests as the system has more cache hits. But the CPU usage once again shoots up to 99%. This needs a deeper analysis and remediation as such risks are often catastrophic.

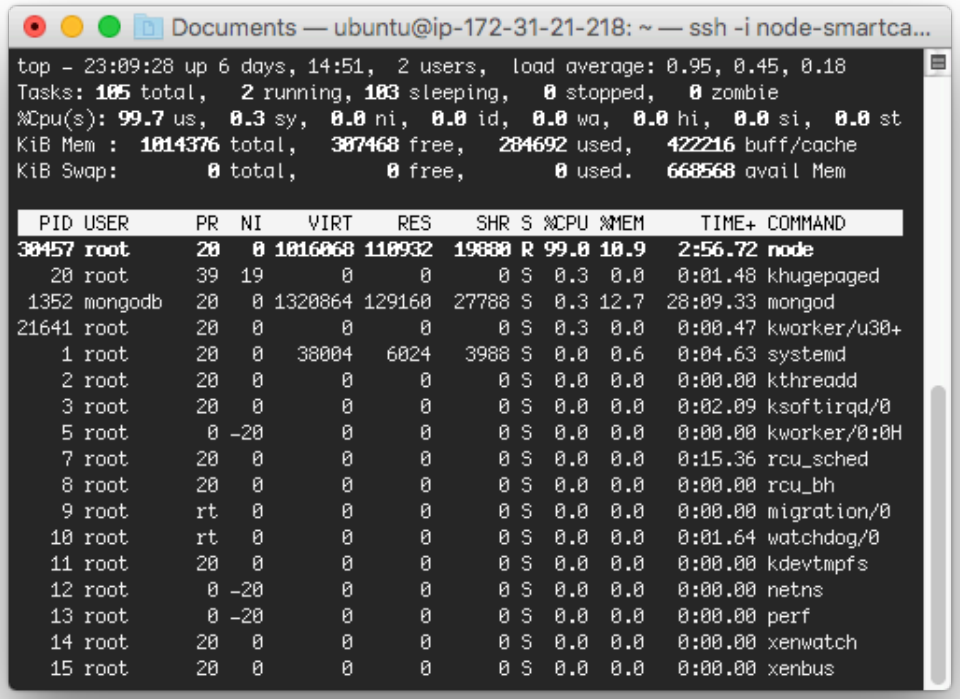


Figure 4.5 Stress Test Server vitals

Chapter 5: Discussion and Conclusions

5.1 Performance Results and Limitations

The iMediCap system helps patients remember to take their medicine on time. It is designed to be as non-intrusive to the user as possible (Bluetooth pairing is the only step required). In addition, now users have an option to use the app without depending on the BLE module. We believe that better adherence to treatments gained by using iMediCap will cut overall long-term healthcare costs and cover the costs of implementing our solution at a large scale.

It can be observed that iMediCap performed adequately well given all the software tools and resources used so far are open source and freely available to public. However, it is obvious that a commercially available form of such system would need to be a lot more dependable in terms of system availability, scalability and usability. But through this report work we have been able to lay a strong foundation for such a system through micro-service design, security related features and performance improvements.

Certain limitations that came to the fore were related to server performance, and graphical design of the mobile and web applications. The server performance can be greatly enhanced by incorporating the Elastic Load Balancing provided by AWS at a nominal cost. Also, AWS provided SSL certificates for apps hosted in EC2, this further improves the security as the communication data over the channels would be encrypted. This again would incur some cost related to purchasing domain etc. The graphics used for the mobile and web applications are quite minimalistic and uses basic HTML 5 and CSS, but by incorporating modern graphics designing tools the UI experience can be significantly improved.

The iMediCap project will be made open source and anyone can make contribution after May 2017. The source code for now can be forked from below repositories.

Mobile App: https://abhibroto0402@bitbucket.org/abhibroto0402/smartcap_app.git

Server Code: https://abhibroto0402@bitbucket.org/abhibroto0402/smartcap-server.git

Perf-Test: https://abhibroto0402@bitbucket.org/abhibroto0402/smartcap-perf-test.git

5.2 FUTURE WORK

Most of the future work would be around improving the UI/UX of the mobile and web application. Additionally, by obtaining a domain we can incorporate SSL secure feature and the elastic load balancing. This would help the iMediCap app to be launched at least as a Beta on the play store.

Appendix

SmartCap 1.0

1.0 ABSTRACT: WHAT IS SMARTCAP?

50% of patients are not taking their medication as prescribed. Medical non-adherence is a multi-billion-dollar industry. As such, we are designing SmartCap: A smart bottle lid designed to fit over existing prescription bottles. The solution will strive to be low cost (<$5 at high volume production), recyclable/disposable, and ubiquitous.

The SmartCap system helps the patient by:

- Acting as a personal assistant on the patient's smartphone, sending discreet medication reminders and taking act on if a medication was not taken at the appropriate time. If the treatment requires multiple medications, SmartCap can propose the optimum schedule.
- Monitoring the storage temperature and providing an alert if the medication has been exposed to extreme temperatures (e.g. a hot car) that could compromise the integrity of the medication and result in undesirable side effects or an ineffective treatment.
- Electronically transferring the prescribed dosage information from the pharmacy that filled the prescription and syncing it with the patient's smartphone through the cloud service. The patient is not required to manually enter any prescription information.

There are three major components to the SmartCap solution: The SmartCap itself, the accompanying Android application, and the pharmacy's server. The following section discusses our envisioned solution, SmartCap, and offers a system overview and a rather casual use case format. The design and requirements section describes the high-level functionality of the system and sets clear requirements on how the system should operate. The actual technical implementation details are left up to the implementation section.

The prototyping process had time limitations and additional functionality that will be included in future releases is discussed in the section 5.0 Difficulties, Limitations, and Future Work. In that section, we take the time to discuss difficulties found during the prototyping process and how we addressed them.

2.0 SMARTCAP'S COMPETITION

Current state of the art medical adherence products does not compare to SmartCap. Some products are not "smart" and others are costly, not portable, or difficult to set up. We are differentiating by making SmartCap cheap, portable, ready for mass production, and non-intrusive to the user.

There are currently two camps of state of the art in medical adherence solutions: ones that focus on high-end pill boxes and those that are focused on low cost, mass-produced cap only solutions. The state of the art full pill box was designed by graduate students at MIT and is named "uBox". It is a relatively large box that fits in the palm of your hand, can be locked, and is used as a medicine dispenser. The MIT graduate students founded a company, AbioGenix, which is making the uBox available to patients by a subscription service of $25 per month per medication.

The other state of the art high-end product is made by a startup called Adheretech. This product originates from work by Dr. Emil Jovanov from the University of Alabama in Huntsville. This design focuses on ease of use to the patient, has a built-in 3G radio, and due to its extremely high cost, can only be used to track adherence of high-cost specialty medication.

There is currently a gap in the low-end segment of the adherence market. This market segment would be for ultra-low cost, disposable, cap-only solutions that provide most if not all the features of the high-end solutions at a fraction of the cost. I will describe each of the existing solutions and indicate the features that would make SmartCap the new state of the art in low-cost, mass-produced adherence solutions.

eCap, by informationmediary.com. Like SmartCap, this product is small and ubiquitous and fits over existing medicine bottles. Unlike SmartCap, this product does not connect to

the Cloud and requires that the medication schedule be manually entered software on the PC. Also, due to the use of NFC for wireless communication, the range is limited to a few cm. SmartCap overcomes the obstacle of manual schedule entry by securely routing this information from the Pharmacy through the cloud and overcomes the range limitation by using BLE.

GlowCap, by vitality.net, uses a WIFI connection from the cap to a dongle which has a built-in GSM radio. The system is bulky and expensive, requiring an initial hardware cost greater than $100 and a monthly subscription fee of $79 per prescription. While this is a cap-only solution, its high cost places it more in the high-end segment. SmartCap targets a $3 to $5 price point and is nearly indistinguishable from a standard medicine cap.

The only product that comes near SmartCap's price point is RxTimerCap. This is a standard child-proof medicine cap with a build-in stopwatch and sells for $3. It does not have any form of wireless communication, does not store a log, and does not measure temperature. The stopwatch is a useful reminder; however, it is not of any use for automatically collecting adherence data. SmartCap logs open/close and temperature change events for the life of the prescription, communicates this information to a cloud server, and automatically obtains the electronic prescription schedule from a cloud service. SmartCap can also send you alerts and be your personal medication assistant.

2.0 ENVISIONED SYSTEM

Our vision for the SmartCap system begins with a casual use case example and moves on to the system overview.

2.1 Use Case

Joe needs to fill a prescription at Walgreens. He approaches the pharmacy and a technician informs him that they have just upgraded their boring prescription bottles to SmartCaps. All Joe needs to do is download the application and sign up and he can easily use any future SmartCap prescriptions!

When Joe gets home he signs up and logs in to the application. This is the first-time Joe has opened the application, so it tells him how the technology works! Joe is surprised that the cap stores all sorts of information from when the bottle is opened or closed to ambient temperature and humidity levels. He
finds it incredible that it syncs to his cell phone automatically. Joe takes his first dose and goes about his day.

The following day Joe receives a text message on his phone telling him that he took his pill yesterday at 3:32 PM and will need to take his second dose soon. It even reminded him that he needs to eat a meal beforehand. It could have even alerted him of any drug interactions if he had multiple prescriptions.

Joe takes the bottle with him as he drives home from work. He later gets a text message on his phone telling him that his prescription is too hot. Oh no! He must have left his prescription in the car. Fortunately, SmartCap reminded him just in time.
Joe will rest easy tonight knowing that none of his personal information is stored on the cap – only a useless random ID number, timestamps, and ambient data. Everything confidential is stored in his cell phone or safely on the pharmacy's servers!

2.2 System Overview

The information in the design and reference section as well as the implementation section is more detailed, so this section can be skimmed.
There are three components to the SmartCap solution (shown below): The smartphone application, the SmartCap itself, and the pharmacy's server.

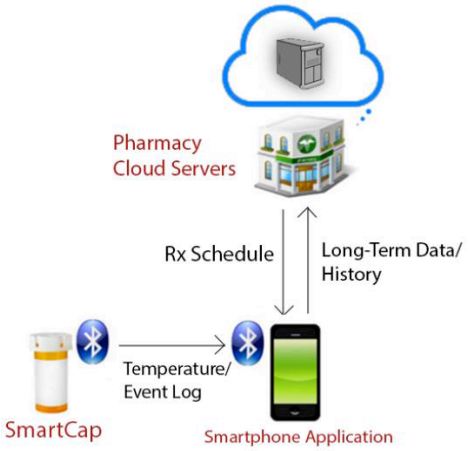


Figure A.1 Components of SmartCap System

2.1.1 The SmartCap Android Application

The application is central to our solution - the application communicates with the SmartCap and the pharmacy's servers: The application is the brain; it provides all the logic.

Communication with the server - A patient registers a username with the pharmacy and logs into the application. The application connects to the server and downloads any new prescription information tied to the user's account. It also sends adherence data to the pharmacy's servers for long-term storage.

Communication with the SmartCap - The application receives open event timestamps and temperature/humidity data from the SmartCap whenever a connection can be established.

Based on the prescription information from the cloud servers, and user activity data from the SmartCap, the application can calculate when the next dose should be taken and send alerts to the user.

Information is stored locally on the phone so the application can continue to function as much as possible without an internet connection. However, the application will store persistent data on the permanent server for robustness.

2.1.2 The SmartCap

The SmartCap communicates solely with the Android application via Bluetooth low energy. The SmartCap is asleep most of the time to conserve energy. Temperature and humidity is polled every 5 minutes and the BLE module only wakes up every 30 minutes or whenever the lid is opened (we wake up the BLE module when the lid is opened because we assume the user has their phone on them when they are taking the medicine and a connection can be established).

Each time a temperature/humidity change or open/close event is generated a data frame consisting of the cap's unique ID, timestamp, event type, and any relevant data measurement is stored in a cyclical buffer. The buffer allows for data to be stored when the phone is not in range. The buffer is emptied and the information is sent to the phone when a connection is established.

2.1.3 The Pharmacy Software and Server

When a prescription is filled, the pharmacy uploads a digital version, along with the SmartCap ID of the bottle, to the patient's cloud account. That way the patient's smartphone can later retrieve all the dose information and calculate dose timing. Each prescription is tied to a different SmartCap unique ID number.

The server also stores permanent user information. When the smartphone application receives event logs from the SmartCap, the phone can forward them to the server for storage.

References

[1] Modular Crypt Format: https://passlib.readthedocs.io/en/stable/modular_crypt_format.html

[2] PassLib Bcrypt: https://passlib.readthedocs.io/en/stable/lib/passlib.hash.bcrypt.html

[3] CanvasJS : http://canvasjs.com/

[4] Mongodb-cron: https://www.npmjs.com/package/mongodb-cron

[5] MongoDB: https://www.mongodb.com/what-is-mongodb

[6] A Secure Cookie Protocol: Alex X. Liu1, Jason M. Kovacs and Mohamed G. Gouda http://www.cse.msu.edu/~alexliu/publications/Cookie/cookie.pdf

[7] CSRF: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[8] CSRF ExpressJS Library: https://www.npmjs.com/package/csrf

[9] CSRF Protection: https://github.com/pillarjs/understanding-csrf

[10] Android activity lifecycle: https://developer.android.com/guide/components/activities/activity-lifecycle.html

[11] http://stackoverflow.com/questions/2550099/how-to-kill-an-android-activity-when-leaving-it-so-that-it-cannot-be-accessed-fr

[12] Elastic Load Balancing: https://aws.amazon.com/elasticloadbalancing/